SD

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/037,666	01/03/2002	Sriram Vajapeyam	42390P11927	8277

8791 7590 08/18/2005

BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 08/18/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)
	10/037,666	VAJAPEYAM ET AL.
	Examiner	Art Unit
	David J. Huisman	2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 June 2005*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-6,8-11,13-15 and 17-30* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-6,8-11,13-15 and 17-30* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *28 December 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *15 June 2005*.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-6, 8-11, 13-15, and 17-30 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: RCE and Amendment as received on 6/9/2005, and IDS as received on 6/15/2005.

Claim Rejections - 35 USC § 102

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

4. Claims 10-11, 13-15, and 17-19 are rejected under 35 U.S.C. 102(b) as being anticipated by Ranganathan et al., "The PEWs Microarchitecture: Reducing Complexity Through Data Dependence-Based Decentralization," 1998 (as disclosed by applicant and herein referred to as Ranganathan).

5. Referring to claim 10, Ranganathan has taught a computer system comprising:

a) at least one memory device to store trace descriptors and instruction sequences. See the 1st paragraph of section 3.3 and note the existence of the trace cache, which holds instruction sequences and information associated with the instruction sequences (trace descriptors).

b) a bus coupled to the at least one memory device. It is inherent a bus is connected to the memory device so data is able to travel between memory and the rest of the system.

c) a control flow logic device to select and fetch one of the trace descriptors, the fetched trace descriptor including a plurality of dependency descriptors having location and dependency information for corresponding instruction sequences.. See Fig.3, the abstract, and section 3.3. Note that when instructions from the trace cache are fetched, the trace descriptor (additional information about the instruction sequence(s)) is also fetched so that it may be used for placing instructions in certain PEWs. The trace descriptor includes a plurality of dependency descriptors which include at least the create map and the use map. These items specify dependency information (which registers are needed by the instructions) and location information (registers are locations).

d) a data flow logic device coupled to the control flow logic device to receive a dependency descriptor dispatched from the control flow logic device, to fetch an instruction sequence corresponding to the received dependency descriptor, and to execute the fetched instruction sequence. See Fig.3, section 3.3, and the abstract. Note that an instruction sequence and corresponding dependency descriptor(s) are fetched from the trace cache. The dependency descriptor is then used to direct the instructions to the appropriate queues for execution.

6. Referring to claim 11, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught an issue window coupled between the control flow logic device and the data flow logic device, the issue window to store the dependency descriptor dispatched from the control flow logic device. See section 3.3 and Fig.3, and note the usher

component. The dependency information is sent (and inherently stored) within the usher so that it may determine how to place the instructions.

7. Referring to claim 13, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that at least one memory unit is to store an instruction sequence contiguously based on dependency information. See Fig.3 (the PEW queues). Instructions are sent to the PEWs based on dependency information and each would be stored until its turn to execute.

8. Referring to claim 14, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught a storage area coupled to the data flow logic device and control flow logic device, the storage area to store live-out data. See Fig.2, and note the ISA-visible register file. Clearly, instructions of a particular group (for instance, those of Fig.5) will be writing to registers within the file. These registers may in turn be used by a subsequent group of instructions.

9. Referring to claim 15, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught a storage area coupled to the control flow logic, the storage area to map live-in and live-out data. See Fig.3, and note the register queues. The register queues, according to section 3.4, implement register-renaming, which is known to be the mapping of live-in and live-out data.

10. Referring to claim 17, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that the fetched trace descriptor includes aggregate live-in data for dependency descriptors in the fetched trace descriptor. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions depend on data which was produced

prior to execution of the given trace. This data is live-in data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-in data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-in data is for a plurality of dependency descriptors.

11. Referring to claim 18, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that the fetched trace descriptor includes aggregate live-out data for dependency descriptors in the fetched trace descriptor. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions produce data which will be depended upon in the future. This data is live-out data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-out data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-out data is for a plurality of dependency descriptors.

12. Referring to claim 19, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that dependency information of the received dependency descriptor includes live-out data. As previously discussed, the dependency descriptor includes data which specifies data dependencies, which in turn means that the data is associated with registers (which represent live-in and live-out values). More specifically, the create map is used to records registers which are modified by the associated sequence (live-out data).

13. Claims 1-6 and 20-21 are rejected under 35 U.S.C. 102(e) as being anticipated by Batten et al., U.S. Patent No. 6,260,189 (as applied in the previous Office Action and herein referred to as Batten).

14. Referring to claim 1, Batten has taught a logic circuit comprising:

a) a control flow logic to select and fetch a trace descriptor for processing, the fetched trace descriptor including at least one dependency descriptor, the dependency descriptor including dependency information for an instruction sequence and a location of the instruction sequence. See Fig.9 and note the ccdd instruction (trace descriptor). Being an instruction, it is inherently fetched. Furthermore, note the format of the trace descriptor in Fig.7. The dtype field and numInstr field make up a dependency descriptor which specifies the types of dependencies found within a following instruction sequence. The numInstr field specifies that the next X number of instructions at the next X number of addresses will have the associated dependencies. Consequently, the numInstr field includes a location for the sequence, where the location is the next X instruction addresses.

b) a data flow logic coupled to the control flow logic to execute the instruction sequence according to the dependency information in the dependency descriptor. See Fig.1 and note that instructions are inherently executed.

15. Referring to claim 2, Batten has taught a logic circuit as described in claim 1. Batten has further taught a storage area coupled to the control flow logic and the data flow logic, the storage area to store the dependency descriptor from the fetched trace descriptor by the control flow

logic. See Fig.7 and note that the ccdd instruction (trace descriptor), which includes the dependency descriptor, is inherently stored.

16. Referring to claim 3, Batten has taught a logic circuit as described in claim 1. Batten has further taught a storage area coupled to the control flow logic, the storage area to store trace descriptors. See Fig.7, and note that the ccdd instruction is inherently stored.

17. Referring to claim 4, Batten has taught a logic circuit as described in claim 1. Batten has further taught a storage area coupled to the data flow logic, the storage area to store instructions contiguously based on dependency information. Looking at Figs.9-12, it can be seen that contiguous instructions in a group correspond to the same dependency information associated with the ccdd instruction. These instructions are inherently stored.

18. Referring to claim 5, Batten has taught a logic circuit as described in claim 1. Batten has further taught a storage area coupled to the data flow logic and control flow logic, the storage area to store live-out data. See Figs.9-12 and note that a register file is present (since instructions access registers). As shown, instructions of a particular group will be writing to registers within the file. These registers may in turn be used by a subsequent group of instructions.

19. Referring to claim 6, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the control flow logic, the storage area to map live-in and live-out data. See Fig.3, and note the register queues. The register queues, according to section 3.4, implement register-renaming, which is known to be the mapping of live-in and live-out data.

20. Referring to claim 20, Batten has taught a method of processing instructions comprising:

a) selecting and fetching a trace descriptor in accordance with program control flow. See Fig.9 and note the ccdd instruction (trace descriptor). Being an instruction, it is inherently fetched.

b) identifying from the fetched trace descriptor a dependency descriptor including dependency information for a set of instructions and a locations of the set of instructions. Note the format of the trace descriptor in Fig.7. The dtype field and numInstr field make up a dependency descriptor which specifies the types of dependencies found within a following instruction sequence. The numInstr field specifies that the next X number of instructions at the next X number of addresses will have the associated dependencies. Consequently, the numInstr field includes a location for the sequence, where the location is the next X instruction addresses.

c) fetching the set of instructions from the location in the dependency descriptor. Again, the location in the dependency descriptor is "the next X addresses". That is, the instruction set may be located and fetched at the next X addresses.

d) executing the set of instructions according to the dependency information in the dependency descriptor. See column 6, lines 44-58, and column 3, lines 52-57. Note that the dependency information is used by the hardware in executing the instructions such that stalls may be avoided by disabling the components which check for stalls.

21. Referring to claim 21, Batten has taught a method as described in claim 20. Batten has further taught updating live-out data in a storage area. It is known that live-out data is data which is generated by a given sequence that is needed by another sequence for input. Looking at Fig.9-10, for instance, assume that the sequence of Fig.10 follows the sequence of Fig.9. It can be seen that r9, s5, and s3, of Fig.9 are live-out values because they are generated by the

sequence of Fig.9 and they are used as inputs by the sequence of Fig.10. And, clearly these values are stored within registers, which are storage.

Claim Rejections - 35 USC § 103

22. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

23. Claims 1-6 and 8-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ranganathan, as applied above, in view of Nair, U.S. Patent No. 6,304,962. Furthermore, Verbauwhede, U.S. Patent No. 5,732,255, is cited as extrinsic evidence for showing a reason why it would be more beneficial to store addresses in a trace cache as opposed to instructions.

24. Referring to claim 1, Ranganathan has taught a logic circuit comprising:

a) a control flow logic to select and fetch a trace descriptor for processing, the fetched trace descriptor including at least one dependency descriptor, the dependency descriptor including dependency information for an instruction sequence. See the first paragraph of section 3.3 and note that data dependence information (dependency descriptor) associated with a particular trace of instructions is stored in the trace cache. The trace descriptor would include all information within the trace cache associated with a particular trace of instructions (associated dependency information, generated bitmaps (section 3.2.2), and any other related information).

b) Ranganathan has taught that an instruction sequence itself is stored in the trace cache. See the last paragraph of section 3.1. Ranganathan has not taught that the dependency descriptor

includes a location (i.e., address) of the instruction sequence. However, Nair has taught a component called a Superblock Target Buffer (STB), which acts like a common trace cache in that it tries to predict long execution paths. However, unlike a common trace cache, the STB stores instruction addresses and not the instructions themselves. See column 10, lines 11-17. Verbauwhede has shown that instruction addresses may be smaller than the instructions themselves. See column 2, lines 43-49. More specifically, Verbauwhede shows an example where an instruction address (location) may be specified by 16 bits where the instructions themselves are 32 bits. A person of ordinary skill in the art would have recognized that by storing the location of the instruction sequence as opposed to the instruction sequence itself, less space would be consumed in the trace cache. In addition, it should be realized that it is not necessarily important whether instructions or instruction addresses (locations) are stored in a trace cache. Instead, all that is required is that instructions may be identified by the trace cache and this may be done by providing the instruction or its address. As a result, it would have been obvious to one or ordinary skill in the art at the time of the invention to modify Ranganathan such that the trace cache (more specifically, the dependency descriptor) stores an instruction location as opposed to the instruction sequence itself.

c) a data flow logic coupled to the control flow logic to execute the instruction sequence according to the dependency information in the dependency descriptor. See the abstract.

25. Referring to claim 2, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the control flow logic and the data flow logic, the storage area to store the dependency descriptor from the fetched trace descriptor by the control flow logic. See section 3.3 and Fig.3, and note the usher component.

The dependency information is sent (and inherently stored) within the usher so that it may determine how to place the instructions.

26. Referring to claim 3, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the control flow logic, the storage area to store trace descriptors. See Fig.3 (trace cache).

27. Referring to claim 4, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the data flow logic, the storage area to store instructions contiguously based on dependency information. See Fig.3 (the PEW queues). Instructions are sent to the PEWs based on dependency information and each would be stored until its turn to execute.

28. Referring to claim 5, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the data flow logic and control flow logic, the storage area to store live-out data. See Fig.2, and note the ISA-visible register file. Clearly, instructions of a particular group (for instance, those of Fig.5) will be writing to registers within the file. These registers may in turn be used by a subsequent group of instructions.

29. Referring to claim 6, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a storage area coupled to the control flow logic, the storage area to map live-in and live-out data. See Fig.3, and note the register queues. The register queues, according to section 3.4, implement register-renaming, which is known to be the mapping of live-in and live-out data.

30. Referring to claim 8, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught that the trace descriptor includes aggregate live-in data for a plurality of dependency descriptors in the trace descriptor. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions depend on data which was produced prior to execution of the given trace. This data is live-in data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-in data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-in data is for a plurality of dependency descriptors.

31. Referring to claim 9, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught that the trace descriptor includes aggregate live-out data for a plurality of dependency descriptors in the trace descriptor. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions produce data which will be depended upon in the future. This data is live-out data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-out data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-out data is for a plurality of dependency descriptors.

32. Claims 22 and 28-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above.

33. Referring to claim 22, Batten has taught a method as described in claim 20.

a) Batten has not taught storing the identified dependency descriptor from a control flow logic into a storage area. However, Official Notice is taken that reservation stations are well known and accepted in the art. A reservation station is additional storage for holding instructions after they have been fetched until the resources are available to execute them. Since the dependency descriptor is part of a ccdd instruction, and instructions are stored in a reservation station, then it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include a reservation station for holding the dependency descriptor until resources are available.

b) reading the dependency descriptor out of the storage area into the data flow logic. Ultimately, the instruction needs to be executed so it will be read from the reservation station.

34. Referring to claim 28, Batten has taught a machine-readable medium that provides instructions, which when executed by a machine cause the machine to perform operations comprising:

a) selecting and fetching a trace descriptor in accordance with program control flow. See Fig.9 and note the ccdd instruction (trace descriptor). Being an instruction, it is inherently fetched.

b) identifying from the fetched trace descriptor a dependency descriptor including dependency information for a set of instructions and a locations of the set of instructions. Note the format of the trace descriptor in Fig.7. The dtype field and numInstr field make up a dependency descriptor which specifies the types of dependencies found within a following instruction

sequence. The numInstr field specifies that the next X number of instructions at the next X number of addresses will have the associated dependencies. Consequently, the numInstr field includes a location for the sequence, where the location is the next X instruction addresses.

c) Batten has not taught storing the dependency descriptor in an issue window to await assignment to an execution unit. However, Official Notice is taken that reservation stations are well known and accepted in the art. A reservation station is additional storage for holding instructions after they have been fetched until the resources are available to execute them. Since the dependency descriptor is part of a ccdd instruction, and instructions are stored in a reservation station, then it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include a reservation station for holding the dependency descriptor until resources are available.

c) fetching the set of instructions from the location in the dependency descriptor. Instructions described by a dependency descriptor are inherently fetched if they are to be executed. Again, the location in the dependency descriptor is "the next X addresses". That is, the instruction set may be located and fetched at the next X addresses.

d) executing the set of instructions according to the dependency information in the dependency descriptor. See column 6, lines 44-58, and column 3, lines 52-57. Note that the dependency information is used by the hardware in executing the instructions such that stalls may be avoided by disabling the components which check for stalls.

35. Referring to claim 29, Batten has taught a medium as described in claim 28. Batten has further taught that the operations further comprise updating live-out data in a storage area. It is known that live-out data is data which is generated by a given sequence that is needed by another

sequence for input. Looking at Fig.9-10, for instance, assume that the sequence of Fig.10 follows the sequence of Fig.9. It can be seen that r9, s5, and s3, of Fig.9 are live-out values because they are generated by the sequence of Fig.9 and they are used as inputs by the sequence of Fig.10. And, clearly these values are stored within registers, which are storage.

36. Referring to claim 30, Batten has taught a medium as described in claim 28. Batten has further taught reading the dependency descriptor out of the issue window into data flow logic. Clearly, in order to execute the ccdd instruction (which includes the dependency descriptor), it must be read from the reservation station (issue window).

37. Claims 23-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above, and further in view of Arimilli et al., U.S. Patent No. 6,427,204 (as applied in the previous Office Action and herein referred to as Arimilli).

38. Referring to claim 23, Batten has taught a method as described in claim 20. Batten has not taught that the fetching of a set of instructions is completed just in time for execution. However, Arimilli has taught such a concept. See column 3, lines 1-17. Note that Arimilli has taught that this is a more efficient way of fetching because instructions are only delivered when they are actually needed and pipeline bubbles are prevented. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten such that instructions are fetched just-in-time, as taught by Arimilli.

39. Referring to claim 24, Batten in view of Arimilli has taught a method as described in claim 20. Although Batten has not taught that the instructions are out of order, Arimilli has taught such a concept. See column 1, line 61, to column 2, line 6. Note that the use of resources

and efficiency are maximized with out-of-order execution. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include instructions that are out-of-order, as taught by Arimilli.

40. Claims 25-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above, in view of Witt et al., U.S. Patent No. 6,018,798 (as applied in the previous Office Action and herein referred to as Witt).

41. Referring to claim 25, Batten has taught a method as described in claim 21. Batten has not explicitly taught updating the architectural state using the data in the storage area. However, Witt has taught the concept of having a speculative register file (future file 88, Fig.3) and an actual register file (Fig.3, component 102). The speculative register file holds the most current state of the machine (values determined via speculative execution) and by doing this, instructions may be executed speculatively. Once it is determined that instructions are no longer speculative, the speculative results are made architectural results by writing them to the actual register file. See column 12, line 66, to column 13, line 45. This is a known concept in the art. In essence, this scheme allows for speculative execution which is a method of executing instructions before it is known that they should execute (they are predicted to execute). This maximizes efficiency if they indeed were to execute (predicted correctly). As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten such that the architectural state is updated using the data in the speculative storage.

42. Referring to claim 26, Batten in view of Witt has taught a method as described in claim 25. Witt has further taught recovering an earlier architectural state after a misprediction using

data in the storage area. See column 18, lines 54-67, and note that after a misprediction, a previous state is achieved by copying actual values into the future file (so that the speculative values are correct). Consequently, by using this newly written data, the system recovers an earlier architectural state.

43. Claim 27 is rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above, and further in view of Rotenberg et al., "A Trace Cache Microarchitecture and Evaluation," 1998 (as applied in the previous Office Action and herein referred to as Rotenberg).

44. Referring to claim 27, Batten has taught a method as described in claim 20. Batten has not taught that the selecting comprises predicting a next trace descriptor to process. However, Rotenberg has taught such a concept. See page 3, and note the first paragraph under Figure 2. Rotenberg has taught that predicting sequences of traces for the implicit achievement of high branch prediction throughput. As a result, in order to increase branch prediction throughput, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include trace prediction as taught by Rotenberg.

Response to Arguments

45. Applicant's arguments filed on June 9, 2005, have been fully considered but they are not persuasive.

46. Applicant argues on pages 11-12 of the remarks, in substance that:

> "Even assuming arguendo that Ranganathan taught a trace descriptor, Applicant respectfully submits any trace descriptor selected and fetched by the tree-level predictor in Figure 3 of Ranganathan would not have a dependency descriptor because Ranganathan taught in sections 3.1 and 3.2.1, for example, that data flow dependences are generated by the RDFG or accessed

from the trace cache only after any such trace descriptor is selected and fetched by the tree-level predictor."

47. These arguments are not found persuasive for the following reasons:

a) Section 3.3 clearly states that dependence information is stored in the trace cache. When the trace is fetched, then the dependence information would be fetched with it. Fig.3 shows that the RDFG analyzer stores data in the cache. It should be realized that the examiner is referring to the trace cache and not to the predictor. Traces are selected and fetched from the trace cache with the dependence information.

48. Applicant argues on page 12 of the remarks, in substance that:

"More particularly, Applicant respectfully submits Batten did not teach or suggest a dependency descriptor including a location of a set of instructions."

49. These arguments are not found persuasive for the following reasons:

a) The numInstr field indicates a location of the instruction set. More specifically, it specifies that the next X number of instructions at the next X number of addresses will have the associated dependencies. Consequently, the numInstr field includes a location for the sequence, where the location is the next X instruction addresses. The instruction set may be located and fetched at the next X addresses.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH
David J. Huisman
August 11, 2005

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100